INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Parallel GMRES with a multiplicative Schwarz preconditioner

Desire NUENTSA WAKAM – Guy Antoine ATENEKENG KAHOU

N° 7342 — version 2

initial version Juillet 2010 — revised version Septembre 2010

Observation and Modeling for Environmental Sciences

ISSN 0249-6399 ISRN INRIA/RR--7342--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Parallel GMRES with a multiplicative Schwarz preconditioner

Desire NUENTSA WAKAM – * Guy Antoine ATENEKENG KAHOU†

Theme : Observation and Modeling for Environmental Sciences
Équipes-Projets Sage

Rapport de recherche n° 7342 — version 2 — initial version Juillet 2010 — revised version Septembre 2010 — 24 pages

Abstract: In this paper, we present an hybrid solver for linear systems that combines a Krylov subspace method as accelerator with some overlapping domain decomposition method as preconditioner. The preconditioner uses an explicit formulation associated to one iteration of the classical multiplicative Schwarz method. To avoid communications and synchronizations between subdomains, the Newton-basis GMRES implementation is used as accelerator. Thus, it is necessary to divide the computation of the orthonormal basis into two steps: the preconditioned Newton basis is computed then it is orthogonalized. The first step is merely a sequence of matrix-vector products and solutions of linear systems associated to subdomains; we describe the fine-grained parallelism that is used in these kernel operations. The second step uses a parallel implementation of dense QR factorization on the resulted basis.

At each application of the preconditioner operator, local systems associated to the subdomains are solved with some accuracy depending on the global physical problem. We show that this step can be further parallelized with calls to external third-party solvers. To this end, we define two levels of parallelism in the solver: the first level is intended for the computation and the communication across all the subdomains; the second level of parallelism is used inside each subdomain to solve the smaller linear systems induced by the preconditioner.

Numerical experiments are performed on several problems to demonstrate the benefits of such approaches, mainly in terms of global efficiency and numerical robustness.

Key-words: domain decomposition, preconditioning, multiplicative Schwarz, Parallel GMRES, Newton basis, multilevel parallelism

* INRIA, Campus universitaire de Beaulieu, 35042 Rennes Cedex. E-mail : desire.nuentsa_wakam@irisa.fr

† INRIA and LRI, Parc Orsay Université, 91893 Orsay Cedex. E-mail : atenekeng@yahoo.com

GMRES parallèle préconditionné par Schwarz multiplicatif

Résumé : Dans cet article, nous présentons un solveur hybride pour la résolution des systèmes linéaires sur des architectures parallèles. Le solveur associe un accélérateur basée sur les sous-espaces de Krylov à un préconditionneur basé sur une décomposition de domaine. Le préconditionneur est défini à partir d'une formulation explicite correspondant à une itération de Schwarz multiplicatif. L'accélérateur est de type GMRES. Dans le but de réduire la communication entre les sous-domaines, nous utilisons une version parallèle de GMRES qui divise en deux étapes la construction de la base du sous-espace de Krylov associée: La première étape consiste à generer les vecteurs de la base dans un pipeline d'opérations à travers tous les processeurs. Les principales opérations ici sont les produits matrice-vecteur et les résolutions des sous-systèmes linéaires dans les sous-domaines. La deuxième étape permet d'effectuer une factorisation QR parallèle sur une matrice rectangulaire formée par les vecteurs construits précédemment; ceci permet d'obtenir une base orthogonale du sous-espace de Krylov.

Les sous-systèmes issus de l'application du préconditionneur sont résolus à différents niveaux de précisions par une factorisation LU ou ILU, en fonction de la difficulté du problème sous-jacent. De plus, cette étape peut être faite en parallèle via des appels aux solveurs externes. Pour cela, nous utilisons deux niveaux de parallélisme dans notre solveur global. Le premier niveau permet de définir les calculs et les communications à travers les sous-domaines. Le deuxième niveau est définie à l'intérieur de chaque sous-domaine pour la résolution des sous-systèmes issus du préconditionneur.

Plusieurs tests numériques sont effectués pour valider l'efficacité de cette approche, en particulier pour les aspects d'efficacité et de robustesse.

Mots-clés : Decomposition de domaine, preconditionnement, Schwarz multiplicatif, GMRES parallèle, Base de Newton, parallélisme multiniveaux.

1 Introduction

In this paper, we are interested in the parallel computation of the solution of the linear system (1)

$$Ax = b \tag{1}$$

with $A \in \mathbb{R}^{n\times n}$, $x, b \in \mathbb{R}^n$. Over the two past decades, the GMRES iterative method proposed by Saad and Schultz [33] has been proved very successful for this type of systems, particularly when A is a large sparse nonsymmetric matrix. Usually, to be robust, the method solves a preconditioned system (2)

$$M^{-1}Ax = M^{-1}b \quad \text{or} \quad AM^{-1}(Mx) = b \tag{2}$$

where M^{-1} is a preconditioner operator that accelerates the convergence of the iterative method.

On computing environments with a distributed architecture, preconditioners based on domain decomposition are of natural use. Their formulation reduces the global problem to several subproblems, where each subproblem is associated to a subdomain; therefore, one or more subdomains are associated to a node of the parallel computer and the global system is solved by exchanging informations between neighboring subdomains. Generally, in domain decomposition methods, there are two ways of deriving the subdomains : (i) from the underlying physical domain and (ii) from the adjacency graph of the coefficient matrix A. In any of these partitioning techniques, subdomains may overlap. Overlapping domain decomposition approaches are known as Schwarz methods while non-overlapping approaches refer to Schur complement techniques. Here, we are interested in preconditioners based on the first class. Depending on how the global solution is obtained, the Schwarz method is *additive* or *multiplicative* [38, Ch. 1]. The former approach computes the solution of subproblems simultaneously in all subdomains. It is akin to the block Jacobi method; therefore, the additive Schwarz method has a straightforward implementation in a parallel environment [13]. Furthermore, it is often used in conjunction with Schur complement techniques to produce hybrid preconditioners [14, 23, 34].

The multiplicative Schwarz method builds a solution of the global system by alternating successively through all the subdomains; it is therefore similar to the block Gauss-Seidel method on an extended system; Thus, compared to the additive approach, it will theoretically require fewer iterations to converge. However, good efficiency is difficult to obtain in a parallel environment due to the high data dependencies between the subdomains. The traditional approach to overcome this is through graph coloring by associating different colors to neighboring subdomains. Hence, the solution in subdomains of the same color could be updated in parallel[38, Ch. 1]. Recently, a different approach has been proposed [4, 6]. In that work, the authors proposed an explicit formulation associated to one iteration of the multiplicative Schwarz method. This formulation requires that the matrix is partitioned in block diagonal form [5] such that each block has a maximum of two neighbors; from this explicit formula, the residual vector is determined out of the computation of the new approximate global solution, and therefore could be parallelized through sequences of matrix-vector products and local solutions in subdomains.

The first purpose of this paper is to present the parallelism that is obtained when this explicit formulation is used to build a preconditioned Krylov basis for the GMRES method. This is achieved through the Newton basis implementation proposed in [7] and applied in [18, 37]. Hence, the usual inner products and global synchronizations are avoided across all the subdomains and the resulted algorithm leads to a pipeline parallelism. We will refer to this as a *first-level of parallelism* in GMRES. The second and main purpose of our work here

is to further use parallel operations when solving local systems associated to subdomains. To this end, we introduce a *second-level of parallelism* to deal with those subsystems. The first observation is that for systems that arise from non-elliptic operators, the number of subdomains should be small to guarantee the convergence. Consequently, the local systems could be very large and should be solved efficiently to accelerate the convergence. The second observation comes from the architecture of the current parallel computers. These two levels of parallelism enable an efficient usage of the allocated resources by dividing tasks across and inside all the available nodes. This approach has been recently used to enhance scalability of hybrid preconditioners based on additive Schwarz and Schur Complement techniques [22].

The remaining part of this paper is organized as follows. Section 2 recall the explicit formulation of the Multiplicative Schwarz preconditioner. After that, a parallel implementation of the preconditioned Newton-basis GMRES is given. Section 3 provides the second level of parallelism introduced to solve linear systems in subdomains. As those systems should be solved several times with different right hand sides, the natural way is to use a parallel third-party solver based on an (incomplete) LU factorization. The parallelism in these solvers is intended for distributed-memory computers; we discuss the benefits of this approach. In section 4, we provide intensive numerical results that reveal good performance of this parallel hybrid solver. The matrices of tests are taken either from public academic repositories or from industrial test cases. Concluding remarks and future directions of this work are given at the end of the paper.

2 A parallel version of GMRES preconditioned by multiplicative Schwarz

In this section, the explicit formulation of the multiplicative Schwarz method is introduced. Then we show how to use it efficiently as a parallel preconditioner for the GMRES method. A good parallelism is obtained thanks to the use of the Newton-Krylov basis.

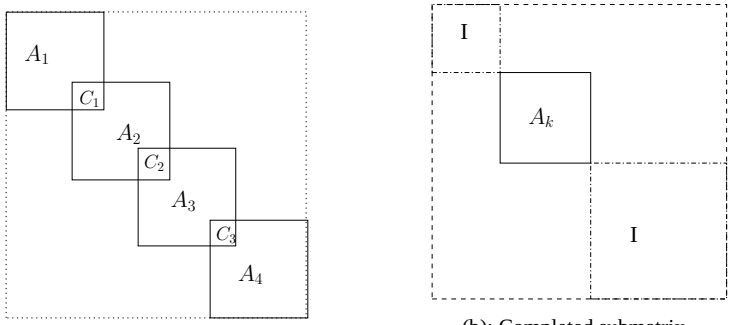
2.1 Explicit formulation of the multiplicative Schwarz preconditioner

From the equation (1), we consider a permutation of the matrix A into p overlapping partitions A_i. We denote by C_i the overlapping matrix between A_i and A_{i+1} $(i = 1, \ldots, p-1)$. Here, each diagonal block has a maximum of two neighbors (see Figure 1.a). Assuming that there is no full line (or column) in the sparse matrix A, such partitioning can be obtained from the adjacency graph of A by means of profile reduction and level sets [5]. We define $\bar{A}_i$ (resp. $\bar{C}_i$) the matrix A_i (resp. C_i) completed by identity to the size of A (see Figure 1.b).

If $\bar{A}_i$ and $\bar{C}_i$ are nonsingular matrices, then the matrix associated to one iteration of the classical multiplicative Schwarz method is defined [6] as :

$$M^{-1} = {\bar{A}_p}^{-1}\bar{C}_{p-1}\bar{A}_{p-1}^{-1}\bar{C}_{p-2}\ldots{\bar{A}_2}^{-1}\bar{C}_1{\bar{A}_1}^{-1} \tag{3}$$

This explicit formulation is very useful to provide stand-alone algorithm for the essential operation $y \leftarrow M^{-1}x$ used in iterative methods. However, since dependencies between subdomains are still present in this expression, no efficient parallel algorithm could be obtained for this single operation. Now, if a sequence of vectors v_i should be generated such that $v_i \leftarrow M^{-1}Av_{i-1}$, then the algorithm would produce a very good pipeline parallelism. For the preconditioned Krylov method, these vectors are simply those that span the Krylov subspace. However, to get a parallel process, the basis of this Krylov subspace should be formed with the Newton polynomials as described in the next section.



(a) : block-diagonal form of A (b): Completed submatrix

Figure 1: Partitioning of A into four subdomains

2.2 Background on GMRES with the Newton basis

A left preconditioned restarted GMRES(m) method minimizes the residual vector $r_m = M^{-1}(b - Ax_m)$ in the Krylov subspace $x_0 + \mathcal{K}_m$ where x_0 is the initial approximation, x_m the current iterate and $\mathcal{K}_m = span\{r_0, M^{-1}Ar_0, \ldots, (M^{-1}A)^m r_0\}$. The new approximation is of the form $x_m = x_0 + V_m y_m$ where y_m minimizes $||r_m||$. The most time consuming part in this method is the construction of the orthonormal basis V_m of $\mathcal{K}_m$; the Arnoldi process is generally used for this purpose [32]. It constructs the basis and orthogonalizes it in the same time. The effect of this is the presence of global communication between all the processes. Hence, no parallelism could be obtained across the subdomains with this approach. However, synchronisation points can be avoided by decoupling the construction of V_m into two independent phases : first the basis is generated *a priori* then it is orthogonalized.

Many authors proposed different ways to generate this *a priori* basis [7, 40, 16]. For stability purposes, we choose the Newton basis introduced by Bai et al [7] and used by Erhel [18]:

$$\widehat{V}_{m+1} = [\mu_0 r_0, \mu_1(M^{-1}A - \lambda_j I)r_0, \ldots, \mu_m \prod_{j=1}^{m}(M^{-1}A - \lambda_j I)r_0] \tag{4}$$

where μ_j and λ_j $(j = 1, \ldots, n)$ are respectively scaling factors and approximate eigenvalues of A. With a chosen initial vector $v_0 = r_0/||r_0||$, a sequence of vectors $v_1, v_2, \ldots, v_m$ is first generated such that:

$$v_j = \sigma_j(M^{-1}A - \lambda_1 I)v_{j-1} \quad j = 1 \ldots m \tag{5}$$

where

$$\sigma_j = 1/||(M^{-1}A - \lambda_j I)v_{j-1}|| \tag{6}$$

To avoid global communication, the vectors v_j are normalized at the end of the process. Hence, the scalars μ_j from the equation (4) are easily computed as the product of scalars σ_j. These steps are explained in detail in the section 2.2.1 and 2.2.2. At this point, we get a normalized basis V_m such that

$$M^{-1}AV_m = V_{m+1}T_m \tag{7}$$

where T_m is a bidiagonal matrix with the scalars σ_j and λ_j. Therefore, V_m should be orthogonalized using a QR factorization :

$$V_{m+1} = Q_{m+1}R_{m+1} \tag{8}$$

As we show in section 2.2.1 and following the distribution of vectors in Figure 3.(b), the v_is are distributed in blocks of consecutive rows between all the processors; However, their overlapped regions are not duplicated between neighboring processors. Thus, to perform the QR factorization, we use a an algorithm introduced by Sameh [35] with a parallel implementation provided by Sidje [37]. Recently, a new approach called TSQR has been proposed by Demmel et al. [17] which aims primarily to minimize the communications and better use the BLAS kernel operations. Their current algorithm used in [27] is implemented with POSIX threads and could only be used when a whole matrix V_m is available in the memory of one SMP node. Nevertheless, the algorithm is SPMD-style and should be easily ported to distributed memory nodes.

So far, at the end of the factorization, we get an orthogonal basis Q_{m+1} implicitly represented as a set of orthogonal reflectors. The matrix R_{m+1} is available in the memory of the last processor. To perform the minimization step in GMRES, we derive an Arnoldi-like relation [32, Section 6.3] using equation (7) and (8)

$$M^{-1}AV_m = Q_{m+1}R_{m+1}T_m = Q_{m+1}\bar{G}_m \tag{9}$$

Hence the matrix $\bar{G}_m$ is in Hessenberg form and the new approximate solution is given by $x_m = x_0 + V_m y_m$ where the vector y minimizes the function J defined by

$$J(y) = ||\beta e_1 - \bar{G}_m y_m||, \quad \beta = ||r_0|| \quad e_1 = [1, 0, \ldots, 0]^T \tag{10}$$

The matrix $\bar{G}_m$ is in the memory of the last processor. Since $m << n$, this least-square problem is sequentially and easily solved using a QR factorization of G_m. The outline of this GMRES algorithm with the Newton basis can be found in [18].

2.2.1 Parallel processing of the preconditioned Newton basis

In this section, we generate the Krylov vectors $v_j, (j = 0, \ldots, m)$ of V_{m+1} from the equation (5). Consider the partitioning of the Figure 1 with a total of p subdomains. At this point, we assume that the number of processes is equal to the number of subdomains. Thus, each process computes the sequence of vectors $v_j^{(k)} = (M^{-1}A - \lambda_j I)v_{j-1}^{(k)}$ where $v_j^{(k)}$ is the set of block rows from the vector v_j owned by process P_k. The kernel computation reduces to some extent to two major operations $z = Ax$ and $y = M^{-1}z$. For these operations, we consider the matrices and vector distribution on Figures 2 and 3. On each process P_k, the overlapping submatrix is zeroed to yield the matrix B_i in Figure 2. In Figure 3.(b), the distribution for the vector x is plotted. The overlapping parts are repeated on all processes. Hence, for each subvector $x^{(k)}$ on process P_k, $x^{(k)^u}$ and $x^{(k)^d}$ denote respectively the overlapping parts with $x^{(k-1)}$ and $x^{(k+1)}$. The pseudocode for the matrix-vector product $z = Ax$ follows then in Algorithm 1.

Now we consider the matrix distribution in Figure 3.(a) for the second operation $y = M^{-1}z$. According to relation (3), each process k solves locally the linear system $A_k t^{(k)} = z^{(k)}$ for $t^{(k)}$ followed by a product $y^{(k)^d} = C_k t^{(k)^d}$ with the overlapped matrix C_k. However, the process P_k should receive first the overlapping part of $y^{(k-1)^d}$ from the process P_{k-1}. Algorithm 2 describes the application of the preconditioner M^{-1} to a vector z. The form of the M^{-1} operator produces a data dependency between neighboring processes. Hence the computation of a single vector $v_j = (M^{-1}A - \lambda_j I)v_{j-1}$ is sequential overall the processses. However since the v_j are computed one after another, a process can start to compute its own part of the vector v_j even if the whole previous vector v_{j-1} is not available. We refer to this as a *pipeline parallelism* since the vectors v_j are computed across all the processes as in a pipeline. This would not be possible with the Arnoldi process as all the global

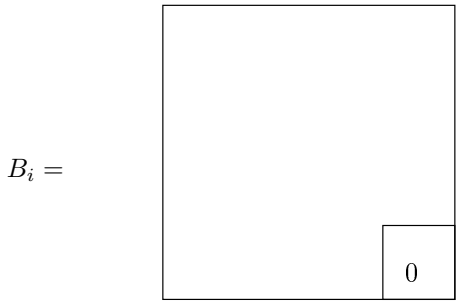
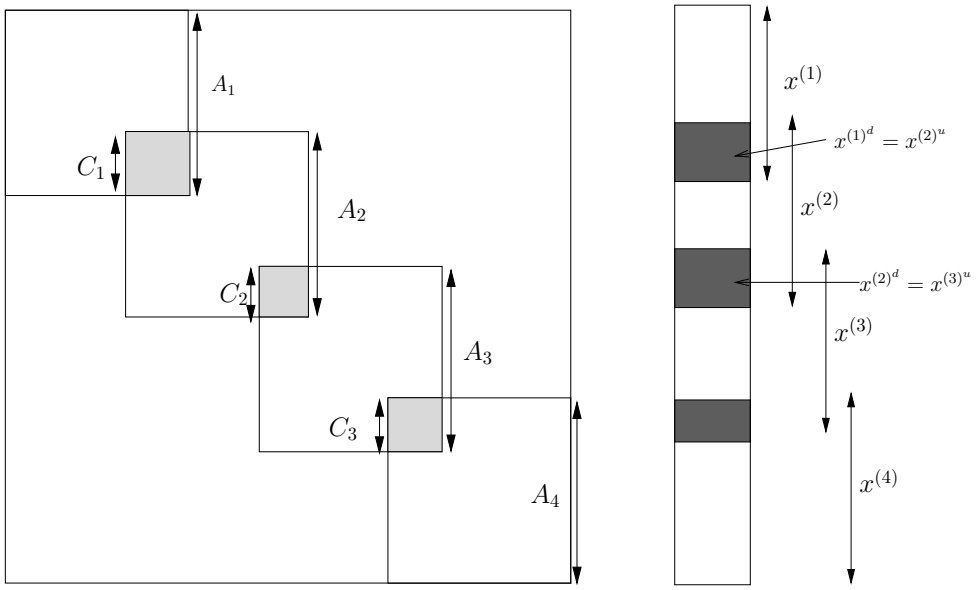


Figure 2: Distribution for the matrix-vector product



(a) : Distribution for the operation $y \leftarrow M^{-1}x$ (b) :Distribution of the vector

Figure 3: Matrix and vector splittings

communications introduce synchronizations points between the processes and prevent the use of the pipeline; see for instance the data dependencies implied by this process in Erhel [18].

To better understand the actual *pipeline parallelism*, we recall here all the dependencies presented in [39]. For the parallel matrix-vector product in Algorithm 1, if we set $z = h(x) = Ax$, then $z^{(k)} = h_k(x^{(k-1)}, x^{(k)}, x^{(k+1)})$ and the Figure 4.(a) illustrates these dependencies. For the preconditioner application $y \leftarrow M^{-1}z$ as written in Algorithm 2, we set $y = g(z) = M^{-1}z$, then $y^{(k)} = g_k(y^{(k-1)}, z^{(k)}, z^{(k+1)})$ and dependencies are depicted on Figure 4.(b). Finally, for the operation $y \leftarrow M^{-1}Ax$, if $y = f(x) = AM^{-1}x = h \circ g(x)$, then $y^{(k)} = f_k(y^{(k-1)}, x^{(k)}, x^{(k+1)}, x^{(k+2)})$ and we combine the two graphs to have the dependencies on Figure 5.a. Hence the dependency $x^{(k-1)}$ is combined with that of $y^{(k-1)}$. In the pipeline flow computation $v_j = f(v_{j-1})$, the dependency $x^{(k+2)}$ in Figure 5.b delays the computation of the $v_j^{(k)}$ until the subvector $v_{j-1}^{(k+2)}$ is available. If there is a gool load balancing between

Algorithm 1 $z = Ax$

1: /* Process P_k holds B_k, $x^{(k)}$ */
2: $k \leftarrow myrank()$
3: $z^{(k)} \leftarrow B_k x^{(k)}$; /* local matrix-vector product */
4: **if** $k < p$ **then**
5: Send $z^{(k)^d}$ to process P_{k+1}
6: **end if**
7: **if** $k > 1$ **then**
8: Receive $z^{(k-1)^d}$ from process P_{k-1}
9: $z^{(k)^u} \leftarrow z^{(k)^u} + z^{(k-1)^d}$
10: **end if**
11: /* Communication for the consistency of overlapped regions */
12: **if** $k > 1$ **then**
13: Send $z^{(k)^u}$ to process P_{k-1}
14: Receive $z^{(k+1)^u}$ from process P_{k+1}
15: $z^{(k)^d} \leftarrow z^{(k+1)^u}$
16: **end if**
17: **return** $z^{(k)}$

Algorithm 2 $y = M^{-1}z$

1: /* Process P_k holds A_k, $z^{(k)}$ */
2: $k \leftarrow myrank()$
3: **if** $k > 1$ **then**
4: Receive $y^{(k-1)^d}$ from process P_{k-1}
5: $z^{(k)^u} \leftarrow y^{(k-1)^d}$
6: **end if**
7: Solve local system $A_k y^{(k)} = z^{(k)}$ for $y^{(k)}$
8: **if** $k < p$ **then**
9: $y^{(k)^d} = C_k y^{(k)^d}$
10: Send $y^{(k)^d}$ to process P_{k+1}
11: **end if**
12: /* Communication for the consistency of overlapped regions */
13: **if** $k > 1$ **then**
14: Send $y^{(k)^u}$ to process P_{k-1}
15: **end if**
16: **if** $k < p$ **then**
17: Receive $y^{(k+1)^u}$ from process P_{k+1}
18: $y^{(k)^d} = y^{(k+1)^u}$
19: **end if**
20: **return** $y^{(k)}$

all the subdomains and if τ denotes a time to compute a subvector $x^{(k)}$ including the communication overhead, then the time to compute one vector is

$$t(1) = p\tau \tag{11}$$

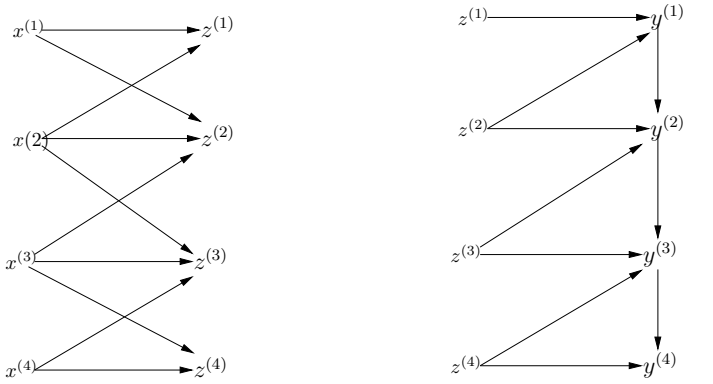
and the time to compute m vectors of the basis follows

$$t(m) = p\tau + 3(m-1)\tau \tag{12}$$

Finally, the first vector is available after $p\tau$ and then, a new vector is produced every 3τ (see Figure 6). The efficiency e_p of the overall algorithm is therefore computed as :

$$e_p = \frac{mt(1)}{pt(m)} = \frac{p\tau m}{p(p\tau + 3(m-1)\tau)} = \frac{m}{p + 3(m-1)} \tag{13}$$

Hence, the efficiency is close to 1/3 for a large value of m. However, this result strongly suggest to keep very small the number p of subdomains, relatively to m.



(a): Graph for the matrix-vector (b): Graph for applying M^{-1}

Figure 4: Dependency graphs

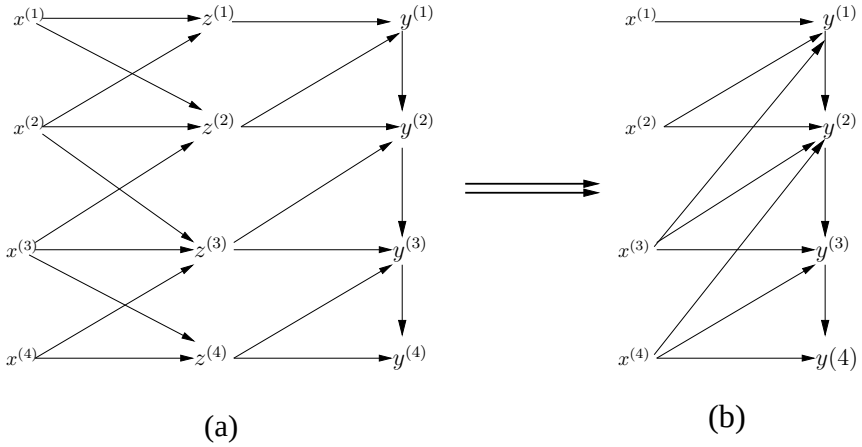


Figure 5: Dependency graph for $y = M^{-1}Ax$

2.2.2 Computation of shifts and scaling factors

So far, we have not yet explained how to compute the shifts λ_i and the scaling factors σ_i of the equation (5) and (6).

As suggested by Reichel [31], the values λ_j should be the m eigenvalues of greatest magnitude of the matrix $M^{-1}A$ in order to have a well-conditioned Krylov basis. Since these

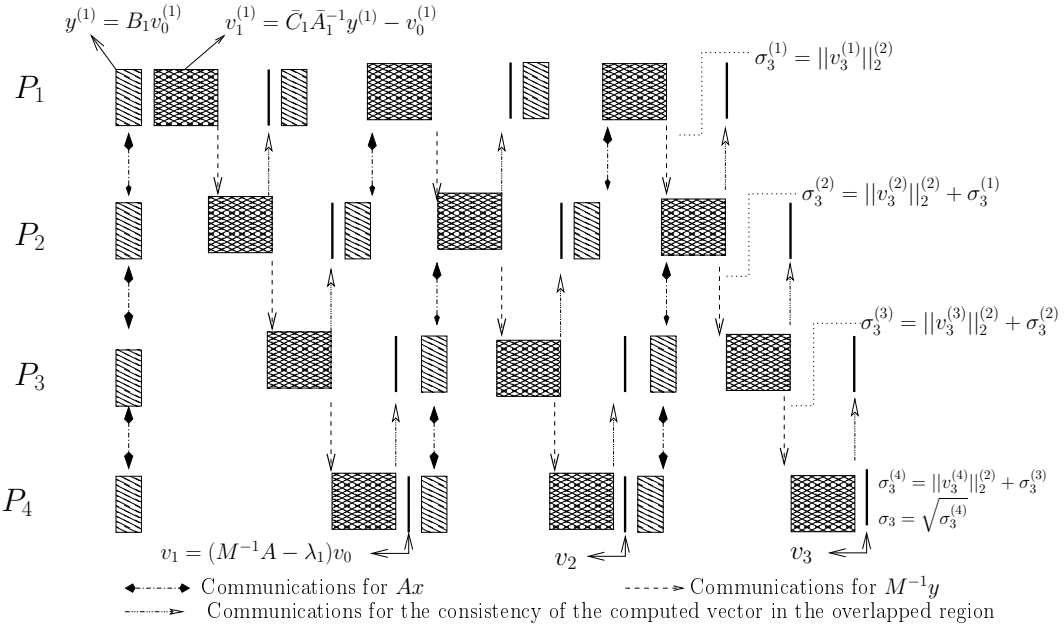


Figure 6: Double recursion during the computation of the Krylov basis

eigenvalues cannot be cheaply computed, we perform one iteration of classical GMRES(m) with the Arnoldi process. Hence, the m eigenvalues computed from the output Hessenberg matrix are the approximate eigenvalues of the preconditioned global matrix [33, Section 3]. These eigenvalues are arranged using the Leja ordering while grouping together two complex conjugate eigenvalues. Recently, Philippe and Reichel [29] proved that, in some cases, roots of the Chebychev polynomials can be used efficiently.

The scalars σ_i are determined without using global reduction operations. Indeed, such operations introduce synchronisation points between all the processes and consequently destroy the pipeline parallelism. The Algorithms 1 and 2 compute in some extent the sequence $v_j = (M^{-1}A - \lambda_i I)v_{j-1}$. We are interested in the scalars $\sigma_j = ||v_j||_2$. For this purpose, on process P_k, we define by $\hat{v}_j^{(k)}$ the subvector $v_j^{(k)}$ without the overlapping part. In the Algorithm 2, after the line 8, an instruction is therefore added to compute the local sum $\sigma_j^{(k)} = ||\hat{v}_j^{(k)}||_2^2$. The result is sent to the process P_{k+1} at the same time as the overlapping subvector at line 10. The next process P_{k+1} receives the result and adds it to its own contribution. This is repeated until the last process P_p. Then at this step, the scalar $\sigma_j^{(p)} = \sqrt{\sigma_j^{(p)}}$ is the norm $||v_j||_2$. An illustration is given in the Figure 6 during the computation of the vector v_3.

3 Parallel computation of local systems in subdomains

During each application of the operator M^{-1}, several linear systems $A_k y^{(k)} = z^{(k)}$ should be solved for $y^{(k)}$ with different values of $z^{(k)}$ (see Algorithm 2 line 7). The matrices $A_k, k = (1 \ldots p)$ are sparse and nonsymmetric as the global matrix A. Hence, during the whole iterative process and inside each subdomain, a sparse linear system with different right hand sides is being solved. On nowadays supercomputers with SMP (Symmetric MultiProcessing)

nodes and with the availability of numerous parallel third-party solvers, it becomes also natural to introduce another level of parallelism within each subdomain. So, in this section, we give first the motivations of using this second level of parallelism. Then we discuss about the appropriate solver to use in subdomains. Finally we give the main steps of our parallel solver with these two levels of parallelism.

3.1 Motivations for two levels of parallelism

In domain decomposition methods, the classical approach is to assign one or several subdomains to a unique processor. In terms of numerical convergence and parallel performance, this approach has some limitations, as pointed out in [22, 24]:

- For some difficult problems, such as those arising from non-elliptic operators, the number of iterations tends to increase with the number of subdomains. It is therefore essential to keep this number of subdomains small in order to provide a robust iterative method. Moreover, with a fixed size m of Krylov basis, a better efficiency is obtained with a small number of subdomains. By doing this, the subsystems are getting large and could not be solved efficiently with only one processor in a subdomain. In this situation, the second level of parallelism is improving the numerical convergence of the method while providing fast and effective solutions in subdomains. On the other hand, the equation 13 suggests, with a fixed size m of the Krylov basis to use a small number of subdomains.
- In a wide range of parallel supercomputers, each compute node is made of several processors accessing more or less the same global memory space (see Figure 7.a). Usually, this space is logically divided by the resource manager if only a subset of processors is scheduled. As a result, the allocated space could not be sufficient to handle all the data associated to a subdomain; it is therefore necessary to schedule for more processors in a node to access the amount of required memory. In this situation, with the one level of parallelism, only one processor will be working (The processor P_0 for instance in Figure 7.b). However, if a new level of parallelism is introduced inside each scheduled node, all the processors could contribute to the treatment of the data stored into the memory of the node(Figure 7.c). Moreover, with this approach, data associated to a particular subdomain could be distributed on more than one SMP node; In this case, the new level of parallelism is defined across all the nodes responsible for a subdomain (Figure 7.d).

3.2 Local solver in subdomains : Accuracy and parallelism issues

The sparse linear systems associated to subdomains are in the scope of the preconditioner operator M^{-1}. Therefore, the solutions of these systems can be obtained more or less accurately:

- An accurate solution, with a direct solver for instance, leads to a powerful preconditioner, thus accelerating the convergence of the global iterative method. This approach is usually known as hybrid direct/iterative technique. The drawback here is the significant memory needed in each subdomain. Indeed, not only the local submatrix A_k should be stored but also the L_k and U_k matrices from its LU factorization. With the presence of fill-in during numerical factorization, the size of those factors can increase significantly compared to the size of the original submatrix.

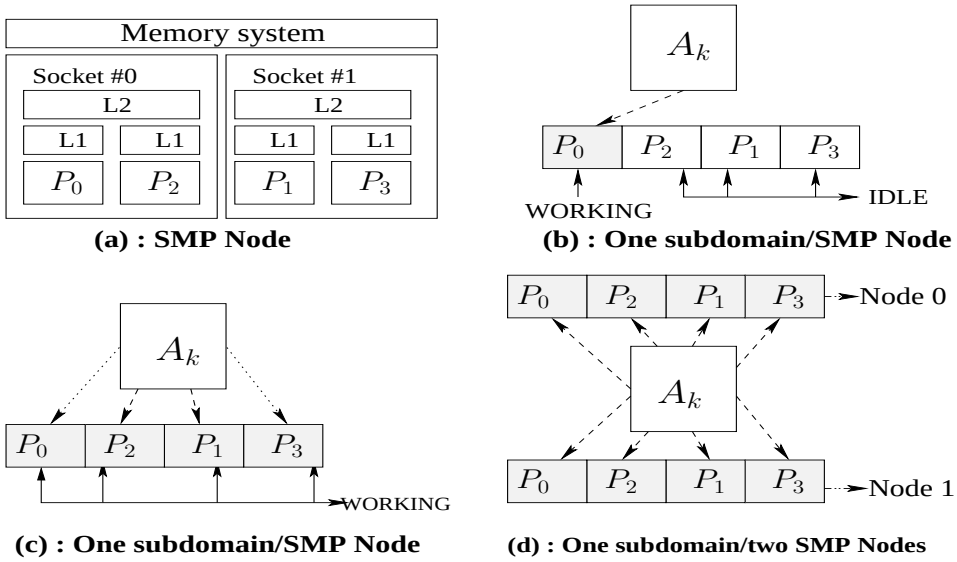


Figure 7: Distribution of subdomains on SMP nodes

- The problem of memory can be overcome by using an incomplete (ILU) factorization of local matrices, thus producing fast solutions in subdomains with fair accuracy. In this case, the global iterative method is more likely to stagnate for some complex problems. An alternative could be to used an iterative method inside each subdomain. However, a large difference in the accuracy of local solutions could change the expression of the preconditioner between external cycles of GMRES.

So far, only the numerical accuracy and the memory issues have being discussed but not the underlying parallel programming model. Beforehand, we assume that a parallel direct solver is used within each computing node when there is enough memory, otherwise an incomplete solver is applied. On SMP nodes with processors sharing the same memory, solvers that distribute tasks among computing units by using thread model are of natural use (Spooles [3], PaStiX [25], PARDISO [36]). On the other side, high performant sparse solvers that use the message-passing model are intended primarily for distributed memory processors. Nevertheless, on SMP nodes, this approach can be used efficiently with the availability of intranode communication channels for the underlying message passing interface (MPI) [12, 21]. These channels are appropriate for the large amount of communications introduced between processors during the solution in subdomains; for instance the comparative study in [2, Sect. 5] give a volume and size of these communications for some direct solvers. As the exchanged messages are of small size, a good data transfer can be obtained inside SMP nodes [11, 30]. Moreover, it is possible to split a subdomain on more than one SMP node. For all these reasons, we use primarily distributed direct solvers in subdomains for the second level of parallelism.

3.3 Main steps of the parallel solver

The solver is composed of the same steps whether a complete or incomplete factorization is used for the subsystems:

1. ***Initialization :*** During the first step, the global matrix A is permuted in block diagonal form by the host process (see Figure 1). After that, the local matrices A_k and C_k should be distributed to other processes. If a distributed solver is used in subdomains, then the processors are dispatched in multiple communicators. For instance, the Figure 8 shows a distribution of four SMP nodes with four processes around two levels of MPI communicators. The first level is intended for the communication across the subdomains. Hence in this communicator, the submatrices are distributed to the *level* 0 processes (i.e $P_{k,0}$ where $k = 0 \ldots p-1$ and p the number of subdomains). Then for each subdomain k, a second communicator is created between the *level* 1 processes to manage communications inside the subdomains (i.e $P_{k,j}$ where $j = 0 \ldots p_k - 1$ and p_k the number of processes in the subdomain k).

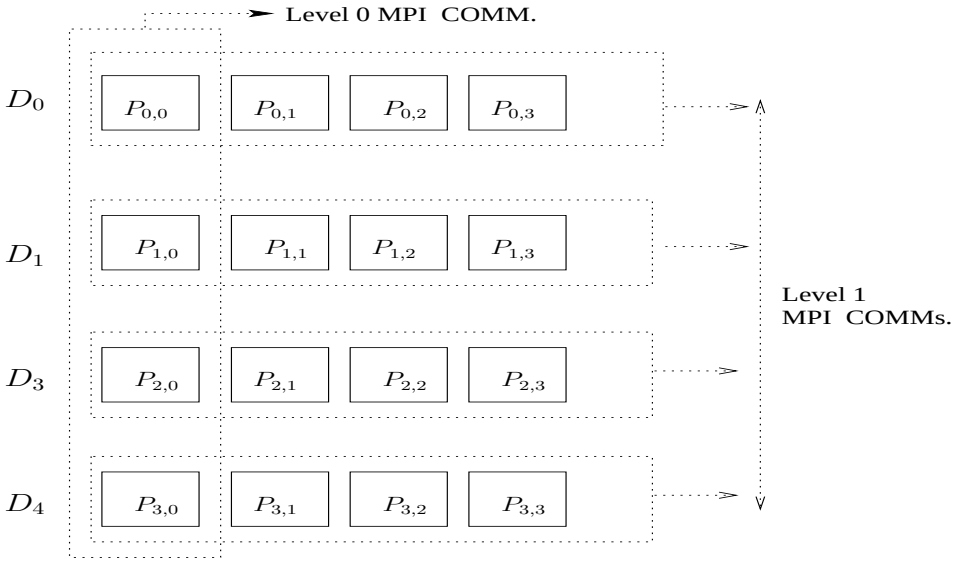


Figure 8: MPI communicators for the two levels of parallelism

2. ***Setup :*** In this phase, the symbolic and numerical factorization are performed on submatrices A_k by the underlying local solver. This step is purely parallel across all the subdomains. At the end of this phase, the factors L_k and U_k reside in the processors responsible for the subdomain k. This is totally managed by the underlying local solver. Prior to this phase, a preprocessing step can be performed to scale the elements of the matrix.

3. ***Solve :*** This is the iterative phase of the solver. With an initial guess x_0 and the shifts λ_j, the solver computes all the equations (5-10) as outlined here:

   (a) Pipeline computation of V_{m+1}

   (b) Parallel QR factorization $V_{m+1} = Q_{m+1}R_{m+1}$

   (c) Sequential computation of $\bar{G}_m$ such that $M^{-1}AV_m = Q_{m+1}\bar{G}_m$ on the process $P_{p-1,0}$.

   (d) Sequential solution of least-square problem (10) for y_m on the process $P_{p-1,0}$.

   (e) Broadcast the vector y_m to processes $P_{k,0}$

(f) Paralle computation of $x_m = x_0 + V_m y_m$

The convergence is reached when $||b - Ax|| < \epsilon ||b||$ otherwise the process restarts with $x_0 = x_m$. When two levels of parallelism are used during the computation of V_m, *level* 1 processors perform multiple backward and forward sweeps in parallel to solve local systems. After the parallel QR factorization in step 3b, the explicit Q factor is never formed explicitly. Indeed, only the unfactored basis V is used to apply the new correction as shown in step 3f. Nevertheless, a routine to form this factor is provided in the solver for general purposes.

4 Numerical experiments

In this section, we perform several experiments to demonstrate the parallel performance and the numerical robustness of our parallel solver on a wide range of real problems. In section 4.1, we give first the software and the hardware architecture on which the tests are done. The sections 4.3 and 4.4 provide intensive results in terms of scalability and robustness on several tests cases. Those cases listed in Table 1 are taken from various sources, ranging from public repositories to industrial softwares

4.1 Software and hardware framework

The solver is named GPREMS[1] (GMRES PRECONDITIONED BY MULTIPICATIVE SCHWARZ). It is intended to be deployed on distributed memory computers that communicate through message passing (MPI). The parallelism in subdomains is based either on message passing or threads model depending on the underlying solver. The whole library is built on top of PETSc (Portable, Extensible Toolkit for Scientific Computation) [8, 9]. The motivation of this choice is the uniform access to a wide range of external packages for the linear systems in subdomain. Moreover, the PETSc package provide several optimized functions and data structures to manipulate the distributed matrices and vectors. A Figure 9 gives the position

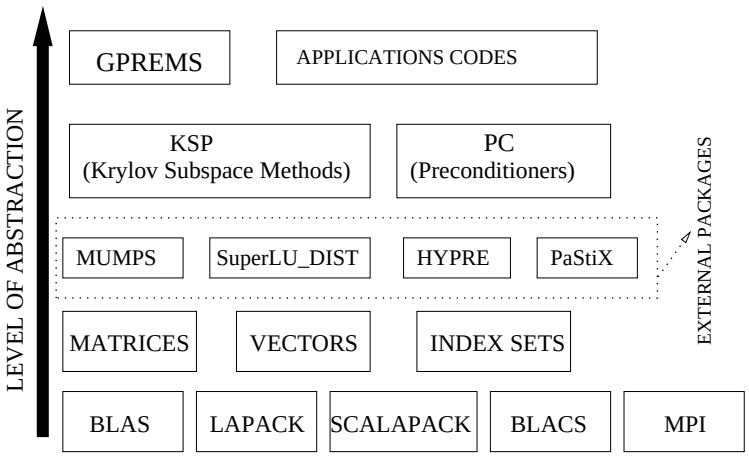


Figure 9: GPREMS library in PETSc

of GPREMS in the abstraction layer of PETSc. We give only the components that are used

[1] A public license will be released soon

by GPREMS. For a complete reference on PETSc environment, please refer to [8]. Although GPREMS uses the PETSc environment, it is not distributed as a part of PETSc libraries. Nevertheless, the library is configured easily once a usable version of PETSc is available on the targeted architecture.

All the tests in this paper are performed on the IBM p575 SMP nodes connected through the Infiniband DDR network. Each node is composed of 32 Power6 processors sharing the same L3 cache. A Power6 processor is a dual-core 2-way SMT with a peak frequency at 4.7 GHz. Figure 10 (obtained with *hwloc*[10]) shows the topology of the first eight processors in one compute node. A total of 128 nodes is available in this supercomputer (named *Vargas*[2]) which is part of the French CNRS-IDRIS computing facilities.

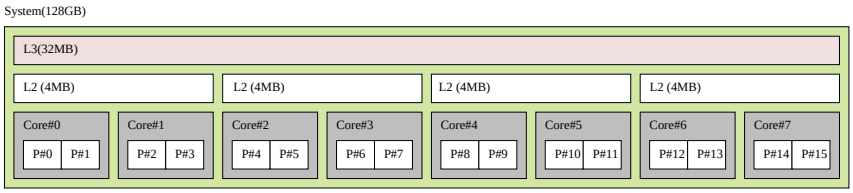


Figure 10: Vargas Architecture

4.2 Test matrices

This section gives the main characteristics of the set of problems under study.

The first set is taken from the University of Florida (UFL) Sparse Matrix Collection [15]. In this set, we choose the problems MEMCHIP and PARA-4 respectively from Freescale and Schenk_ISEI directories; main characteristics of which are given in Table 1. The MEMCHIP

Table 1: General properties of the four test matrices

Matrix Name	MEMCHIP	PARA_4	CASE_004	CASE_017
Order	2,707,524	153,226	7,980	381,689
No. entries (nnz)	14,810,202	2,930,882	430,909	37,464,962
Entries per row	5	19	53	98
Pattern symmetry	91%	100%	88%	93%
2D/3D problem	No	yes	2D	3D
Source	UFL (K. Gullapalli)	UFL (O. Schenk)	FLUOREM	FLUOREM

problem arise from circuit simulation field. The density of the associated matrix is rather small. The number of entries is about $1.4 \cdot 10^6$ but the nonzeros elements per row reveals a rather small density. Hence, this problem is easily solved with GPREMS using an incomplete factorization in subdomains. We present the results with this case in section 4.3. The next matrix is PARA-4 coming from 2D semiconductor device simulations. Compared to the previous problem, the geometry of this problem makes it difficult to solve. Hence, we choose this matrix to show the robustness of our solver when it is combined with a direct solver in subdomains. Again, the results on this are shown in section 4.3.

The second set contains two test matrices (CASE_004 and CASE_017) provided by FLUOREM [20], a fluid dynamics software publisher. In those cases, the sparse matrix corresponds to the global jacobian matrix resulting from the partial first-order derivatives of

[2] http://www.idris.fr/su/Scalaire/vargas/hw-vargas.html

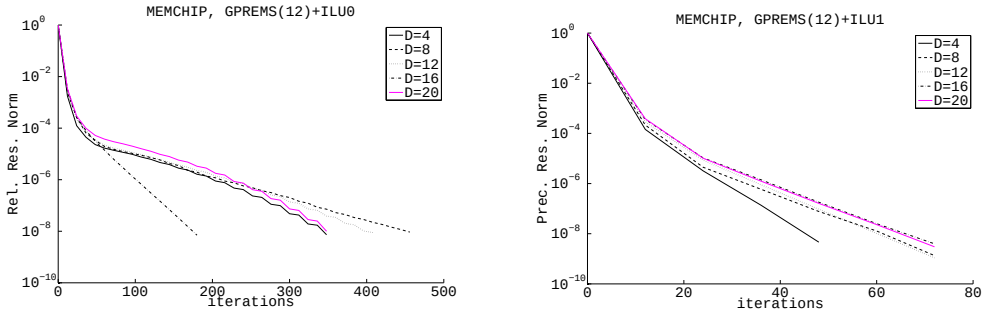
the discrete steady Euler or Reynolds-averaged Navier-Stokes equations. The derivatives are done with respect to the seven conservative fluid variables (mass: ρ, momentum :$[\rho u, \rho v, \rho w]$, energy: ρE, turbulence: $[\rho k, \rho\omega]$ with $k-\omega$); the steady equation can be roughly written as $\mathcal{F}(\mathcal{Q}) = 0$ where Q denotes the fluid variables and $\mathcal{F}(\mathcal{Q})$ the divergence of the flux function. If $\mathcal{Q}_h$ and $\mathcal{F}_h$ are the finite volume discretizations over the computational domain of $\mathcal{Q}$ and $\mathcal{F}$ respectively, then the matrix A of the linear system (1) is the global jacobian matrix $J_{\mathcal{F}_h}(\mathcal{Q}_h)$. $A \in \mathbb{R}^{n\times n}$ is non-symmetric and indefinite and when the stationary flow is dominated by advection, it corresponds to a strongly non-elliptic operator. The two-level parallelism is then useful in those cases in that, we can increase the number of processors to speed up the overall execution time while providing the guarantee that the number of iterations will remain almost constant. Section 4.4 gives the results with the multilevel approach on the CASE_017 case. The matrix CASE_004 is used to show the numerical robustness of GPREMS compared to the additive Schwarz approach implemented in PETSc.

4.3 Convergence and scalability studies on MEMCHIP and PARA-4 cases

We discuss here the results of the numerical convergence of GPREMS on the two problems MEMCHIP and PARA-4. We use respectively $m = 12$ and $m = 24$ as the size of the Newton basis (search directions) for the two cases. Note that the iterative process stops when the relative residual norm $||b - Ax||/||b||$ is less than 10^{-8} or when 1500 iterations are reached. The total number of iterations is usually a multiple of m as the Newton basis should be generated a priori; hence the convergence is tested only at each external cycle. $ILU(p)$ denotes a use of an incomplete factorization in subdomains with p levels of *fill-in* in factored matrix. The package used to perform this factorization is EUCLID[26] from the HYPRE suite [19]. Some other problems like PARA-4 are more difficult to solve. Hence the use of hybrid direct-iterative solver is necessary to achieve a good convergence in a reasonable CPU time. The MUMPS [1] package is then used at this point. On all cases so far, the relative residual norm of the computed solution is given with respect to the number of iterations. We compare the convergence history on 4, 8 and 16 subdomains. Following the convergence history, we give the CPU time spent in the various phases: the initialization phase, the setup phase and the iterative phase.

For the MEMCHIP case with about $1.5 \cdot 10^7$ entries, it takes less than 25 cycles to converge as shown on Figure 11.(a). Moreover, the number of iterations tends to decrease with 16 subdomains. The $ILU(0)$ factorization is used in subdomains meaning that no new elements are allowed during the numerical factorization and thus no more space is needed. This has the effect to speed-up the setup phase and the application of the M^{-1} operator in the iterative step. The CPU times of these steps are shown in Table 2 along with the initialization time (Init) and the total time. These results prove the scalability for almost all the steps in GPREMS. Apart from the initialization step which is sequential, all others steps benefit from the first level of parallelism in the method. The setup phase is fully parallel across the subdomains. Here each process responsible for a subdomain performs a sequential factorization on its local matrix. As the size of those submatrices decreases with the number of subdomains, the setup time decreases as well. The iterative part of the solver (given by *Iter. loop time* in Table 2) scales very well in this case. For instance, 300 iterations of GMRES are performed in less than 103 s. when using 4 subdomains in the block-diagonal partitioning. With 8 subdomains, the method requires 312 iterations (26 restarts or cycles) computed in less than 65 s. When the problem is subdivided into 16 subdomains, the method reveals a superlinear speedup, thanks to the rather small number of iterations needed to converge. This is a particular case probably due to the unpredictable properties

of the incomplete solver in subdomains, the ILU factorization for instance. Indeed, this particular case on 16 subdomains does not appear when $ILU(1)$ is used; see Figure 11.(b).



(a) (b)

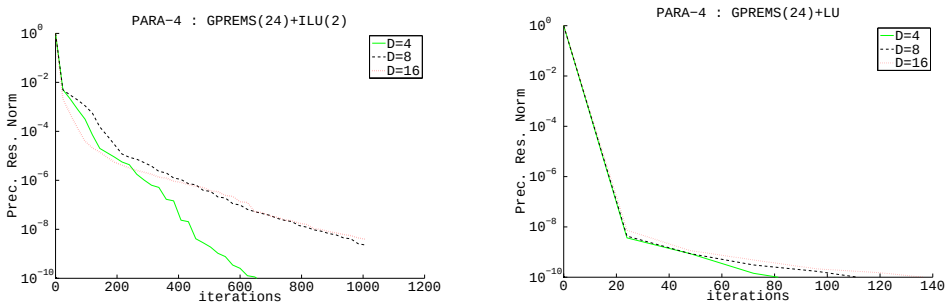
Figure 11: Convergence history with MEMCHIP case on 4, 8 and 16 subdomains

Table 2: MEMCHIP : CPU Time for various numbers of subdomains; ILU(0) is used in subdomains

#Processors	#Subdomains	Iterations	CPU Time (s.)				
			Init	Setup	Iter. loop	Time/Iter	Total
4	4	48	26.38	3.39	33.2	0.69	62.97
8	8	72	27.4	2.03	25.98	0.36	55.41
16	16	60	30.41	1	17.59	0.29	49

Obviously, when the subsystems are solved approximately, the global method looses some robustness. Thus for many 2D/3D cases, the number of iterations gets very large and consequently the overall time needed to converge is too high. We illustrate this behaviour with the PARA-4 2D case. An incomplete LU factorization ($ILU(2)$) is first performed on each submatrix to solve the systems induced by the preconditioner operator. The ILU(2) factorization keeps the second-order fill-ins in the factors, leading to a more accurate preconditioner than that with $ILU(0)$. However, this is not good enough to achieve a good convergence rate as in the previous case. For instance in Figure 12.(a), after reaching 10^{-7}, the residual norm tends to stagnate. This can be observed either with 4, 8 or 16 subdomains. On the other hand, it takes a certain time to setup the block matrices for the iterative phase as shown in Table 3. It can be seen that the maximum time overall processors (*Setup*) to get the incomplete factorization of local matrices is twice than the time of the iterative loop.

Now if a direct solver is used within each subdomain, the method will get a better convergence thanks to accuracy achieved during the application of the preconditioner. This behaviour is observed in Figure 12.(b) in which we show the convergence history for the number of subdomains under study. The number of iterations grows slightly when we change from 4 to 8 subdomains. This does not impact the overall time spent in the iterative step as shown in the second part of the Table 3. Indeed, this step reveals a very good speedup as we only need 35 s. with 8 subdomains compared to 45 s. for 4 subdomains. With 16 processors (subdomains), the total time continues to decrease, even with the slight increase of the global number of iterations. It is also worth pointing out the time spent in the setup phase. It takes roughly 5 s. on four processors to perform the symbolic and the numerical factorization of the submatrices. Note that each process holds a local matrix



(a) (b)

Figure 12: Convergence history with PARA-4 case on 4, 8 and 16 subdomains

associated to one subdomain. Hence, the setup time is the maximum time overall active processors. When the matrix is splitted into 16 subdomains, the size of these submatrices is rather small and the setup time drops to 1 s. This is almost 400 times smaller than that with the ILU(2) factorization. The main reason is the algorithm used by the two methods. In MUMPS, almost all the operations are done on block matrices (the so-called frontal matrices) *via* calls to level 2 and level 3 BLAS functions. Therefore, the different levels of the cache memory are used efficiently during the numerical factorization which is the most time-consuming step. Moreover, a multithreaded BLAS is linked to MUMPS to further speedup the method on SMP nodes. For this reason and the rather small size of submatrices, the second level of parallelism is not applied up to this point.

Table 3: **PARA-4** : CPU Time of GPREMS(24) for various number of subdomains and various types of solvers in subdomains;

#D	#Proc.	Iter.	CPU Time (s.)						
			Init	Setup	Iter. loop	Time/Iter	Total	s_p	e_p
Incomplete solver in subdomains (ILU(2) - EUCLID)									
4	4	1008	1.33	418.17	654.45	0.65	1073.95	-	-
8	8	1008	1.31	409.68	599.64	0.59	1010.63	1.06	0.53
16	16	1008	1.51	398.61	542.75	0.54	942.87	1.14	0.28
Direct solver in subdomains (LU - MUMPS)									
4	4	144	1.32	5.12	45.43	0.32	51.87	-	-
8	8	216	1.26	1.67	35.06	0.16	37.98	1.37	0.68
16	16	240	1.37	0.73	29.21	0.12	31.31	1.66	0.41

In Table 3, we also give the speedup and efficiency of the method relatively to four processors. Clearly, $s_p = T_4/T_p$ and $e_p = 4s_p/p$. We observe that the efficiency is decreasing very fast with the number of subdomains. With ILU(2) in subdomains for instance, the efficiency on 8 and 16 subdomains are respectively 0.53 and 0.28. This behaviour can be explained by the the formula in Equation (13) as p should be keeped small relatively to m. This is one of the reasons to use the second level of parallelism inside the subdomains.

4.4 Benefits of the two-levels parallelism

Now, let us consider the FLUOREM problem CASE_017 listed in Table 1. The geometry of this 3D case is a jet engine compressor. Although the turbulent variables are frozen

during the differentiation, this test case is difficult to solve as shown in a short comparative study involving some distributed linear solvers [28]. From this study, solvers that combine hybrid direct-iterative approaches like GPREMS provide an efficient way to deal with such problems. However, the number of subdomains should be kept small to provide a fast convergence. Therefore the two-levels of parallelism introduced in section 3.2 provide the way to do this efficiently even with a large number of processors. In Table 4, we report the benefit of using this approach on the above-named 3D case. We keep 4 and 8 subdomains and we increase the total number of processors. As a result, almost all the steps in GPREMS get a noticeable speedup. Typically, with 4 subdomains, when only one processor is active, the time to setup the block matrices is almost 203 s. and the time spent in the iterative loop is 622 s. Moving to 8 active processors decreases these times to 59 s. and 181 s. respectively. The same observation can be done when using 8 subdomains, in which case the overall time drops from 540 s. to 238 s. Note that the overall time includes the first sequential step that permutes the matrix in block-diagonal form and distributes the block matrices to all active processors.

The last two columns of Table 4 give the intranode speedup and efficiency of GPREMS. This is different from the ones computed with one level of parallelism. Clearly, we want to show the benefit of adding more processors in a subdomain. Hence, if d is the number of subdomains, $s_p = T_d/T_p$ and $e_p = ds_p/p$ where T_p is the CPU time on p processors. When we add more processors in each subdomain, we observe a noticeable speedup; with 8 subdomains for instance, the speedup moves from 1.18 to 2.28 when 1 and 8 processes are used within each subdomain. However, a look on the efficiency reveals a quite poor scalability. This is a general observation when the tests are done on multicore nodes, when the compute units share the same memory bandwith; indeed, the speed of sparse matrix computations are determined rather by the size of this memory bandwith than the clock rate of CPUs. Nevertheless, the total CPU time to get the solution always suggests to use all the available processors/cores in the compute node; Note that these processors in the node would be idle otherwise as they are scheduled to access more memory.

Table 4: Benefits of the two-levels of parallelism for various phases of GPREMS on CASE_17 with a restart at 64 and MUMPS direct solver in subdomains

#D	#Proc.	Iter.	CPU Time (s.)						
			Init	Setup	Iter. loop	Time/Iter	Total	s_p	e_p
4	4	128	70.78	203.67	622.94	4.87	897.39	-	-
	8	128	70.77	125.77	411.42	3.21	607.96	1.48	0.74
	16	128	69.79	73.15	280.41	2.19	423.35	2.12	0.53
	32	128	70.77	59.44	181.45	1.42	311.66	2.88	0.36
8	8	256	69.91	71.73	399.34	1.56	540.98	-	-
	16	256	70.36	42.99	343.25	1.34	456.59	1.18	0.59
	32	256	71.61	28.72	206.7	0.81	307.02	1.76	0.44
	64	256	71.85	20.85	144.79	0.57	237.49	2.28	0.28

4.5 Numerical robustness of GPREMS

In this section, our aim is to show the robustness of the GPREMS solver on CASE_004 listed in Table 1. The matrix of this 2D test case corresponds to the global jacobian of the discrete steady Euler equation [20]. The two turbulent variables are kept during the differentiation predicting a difficult convergence for any iterative method. The reciprocal

condition number is estimated to be $3.27 \cdot 10^{-6}$. The matrix is permuted into 4 block diagonal matrices producing local matrices with approximately 2,000 rows/columns and 120,000 nonzeros entries. For these relatively small block matrices, we choose an incomplete local solver in subdomains when applying the M^{-1} operator. We also allow 5 degrees of fill-ins ($ILU(5)$) during the incomplete numerical factorization. Figure 13 gives the history of the preconditioned residual norm at each restart for 16 and 32 Krylov vectors. This norm drops to 10^{-11} in less than 200 iterations for GPREMS(16) with ILU(5) in subdomains. When the size of the Krylov subspace varies to 32, the residual vector reaches 10^{-9} in less than 100 iterations. This is almost the same behaviour when a direct solver is used within the subdomains in which case the overall number of iterations is 112 and 96 respectively for 16 and 32 Krylov vectors.

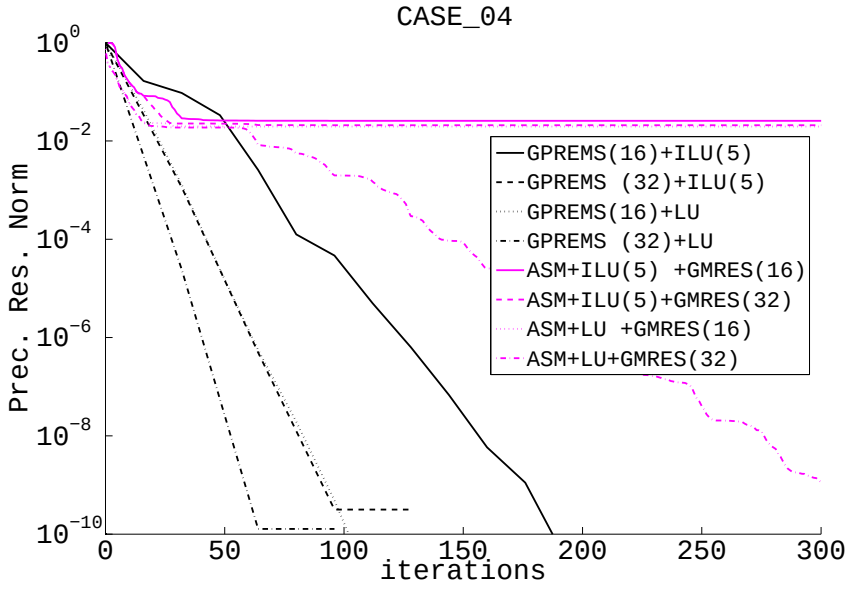


Figure 13: The multiplicative Schwarz approach (GPREMS) compared to the restricted additive Schwarz (ASM) on CASE_004; 16 and 32 vectors in the Krylov subspace at each restart; 4 subdomains (block diagonal partitioning in GPREMS and Parmetis for ASM); local systems solved with different accuracies (ILU(5) and LU)

Then we compare with the restricted additive Schwarz method used as a preconditioner for GMRES. The tests are done with the implementation provided in PETSc release 3.0.0-p2. Here, the Krylov basis is built with the modified Gram-Schmidt (MGS) method and the input matrix is partitioned in 4 subdomains using ParMETIS. In Figure 13, we depict again the preconditioned residual norm with respect to the number of iterations. The curves are given in magenta with the same line specifications as for GPREMS. The main observation is that the iterative method stagnates from the first restart when using ILU(5) in subdomains (the magenta plain and dash curves). With a direct solver, it is necessary to form 32 Krylov vectors at each restart in order to achieve convergence (as shown by the magenta dash-dotted curve).

5 Concluding remarks

In this paper, we give an implementation of a parallel solver for the solution of linear systems on distributed-memory computers. This implementation is based on an hybrid technique that combines a multiplicative Schwarz preconditioner with a GMRES accelerator. A very good efficiency is obtained using a Newton basis GMRES version; hence we define two levels of parallelism during the computation of the orthonormal basis needed by GMRES : The first level is expressed through pipeline operations across all the subdomains. The second level uses a parallelism inside third-party solvers to build the solution of subsystems induced by the domain decomposition. The numerical experiments demonstrate the efficiency of this approach on a wide range of test cases. Indeed, with the first-order parallelism, we show a good efficiency when using either a direct solver or an incomplete *LU* factorization. For the second level used within the subdomains, the gain obtained is two-fold: the convergence is guaranteed when the number of processors grows as the number of subdomains remains the same. The global efficiency increases as we add more processors in subdomains. Moreover, on some difficult problems, the use of a direct solver in subdomains implies to access the whole memory of a SMP node. Therefore, this approach keeps busy all the processors of the node and enables a good usage of allocated computing resources. Finally, on some test cases we show that this solver can be competitive with respect to other overlapping domain decomposition preconditioners.

However, more work needs to be done to achieve very good scalability on massively parallel computers. Presently, an attempt to increase the number of subdomains up to 32 increases the number of iterations as well. The main reason is the form of the multiplicative Schwarz method as each correction of the residual vector should go through all the subdomains. A first attempt is to use more than two levels of splitting but the efficiency of such approach is not always guaranteed, specifically if the linear system arises from non-elliptic partial differential equations. An ongoing work is to keep the two-levels of splitting for the preconditioner operator and then to further accelerate the GMRES method with spectral informations gathered during the iterative process.

References

[1] Amestoy, P. R., Duff, I. S., L'Excellent, J.-Y., and Koster, J. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications 23*, 1 (2001), 15–41.

[2] Amestoy, P. R., Duff, I. S., L'Excellent, J.-Y., and Li, X. S. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Transactions on Mathematical Software 27* (2000), 2001.

[3] Ashcraft, C., and Grimes, R. Spooles: An object-oriented sparse matrix library. In *Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing* (1999).

[4] Atenekeng-Kahou, G.-A. *Parallélisation de 'GMRES' préconditionné par une itération de Schwarz multiplicatif.* PhD thesis, University of Rennes 1 and University of Yaounde 1, 2008.

[5] Atenekeng-Kahou, G.-A., Grigori, L., and Sosonkina, M. A partitioning algorithm for block-diagonal matrices with overlap. *Parallel Computing 34*, 6-8 (2008), 332–344.

[6] Atenekeng-Kahou, G.-A., Kamgnia, E., and Philippe, B. An explicit formulation of the multiplicative schwarz preconditioner. *Applied Numerical Mathematics 57*, 11-12 (2007), 1197 – 1213.

[7] Bai, Z., Hu, D., and Reichel, L. A Newton basis GMRES implementation. *IMA J Numer Anal 14*, 4 (1994), 563–581.

[8] Balay, S., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H. PETSc users manual. Tech. Rep. ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, 2008.

[9] Balay, S., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H. PETSc Web page, 2009. http://www.mcs.anl.gov/petsc.

[10] Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., and Namyst, R. hwloc: a generic framework for managing hardware affinities in HPC applications. In *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010), Pisa, Italia* (2010).

[11] Buntinas, D., and Mercier, G. Data transfers between processes in an SMP system: performance study and application to MPI. In *Proceedings of the 2006 International Conference on Parallel Processing (ICPP 2006), IEEE Computer Society* (2006), pp. 487–496.

[12] Buntinas, D., Mercier, G., and Gropp, W. Implementation and evaluation of shared-memory communication and synchronization operations in mpich2 using the nemesis communication subsystem. *Parallel Computing 33*, 9 (2007), 634 – 644. Selected Papers from EuroPVM/MPI 2006.

[13] Cai, X.-C., and Saad, Y. Overlapping domain decomposition algorithms for general sparse matrices. *Numerical Linear Algebra with Applications 3*, 3 (1996), 221–237.

[14] Carvalho, L., Giraud, L., and Meurant, G. Local preconditioners for two-level non-overlapping domain decomposition methods. *Numerical Linear Algebra with Applications 8*, 4 (2001), 207–227.

[15] Davis, T. The university of Florida sparse matrix collection, 2010. Submitted to ACM Transactions on Mathematical Software.

[16] de Sturler, E. A parallel variant of GMRES(m). In *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics, Dublin, Ireland* (1991), J. J. H. Miller and R. Vichnevetsky, Eds., Criterion Press.

[17] Demmel, J., Grigori, L., Hoemmen, M., and Langou, J. Communication-optimal parallel and sequential QR and LU factorizations. Tech. Rep. UCB/EECS-2008-89, University of California EECS, May 2008.

[18] Erhel, J. A parallel GMRES version for general sparse matrices. *Electronic Transaction on Numerical Analysis 3* (1995), 160–176.

[19] Falgout, R., and Yang, U. Hypre: a library of high performance preconditioners. In *Lectures Notes in Computer Science* (2002), vol. 2331, Springer-Verlag, pp. 632–641.

[20] FLUOREM. The fluorem matrix collection, 2009. LIB0721 2.0 / FP-SA http://www.fluorem.com.

[21] Geoffray, P., Prylli, L., and Tourancheau, B. Bip-smp: High performance message passing over a cluster of commodity smps. In *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing* (1999).

[22] Giraud, L., Haidar, A., and Pralet, S. Using multiple levels of parallelism to enhance the performance of domain decomposition solvers. *Parallel Computing In Press* (2010).

[23] Giraud, L., Haidar, A., and Watson, L. Parallel scalability study of hybrid preconditioners in three dimensions. *Parallel Computing 34*, 6-8 (2008), 363 – 379.

[24] Haidar, A. *On the parallel scalability of hybrid linear solvers for large 3D problems.* PhD thesis, Institut National Polytechnique de Toulouse, 2008.

[25] Hénon, P., Pellegrini, F., Ramet, P., Roman, J., and Saad, Y. High Performance Complete and Incomplete Factorizations for Very Large Sparse Systems by using Scotch and PaStiX softwares. In *Eleventh SIAM Conference on Parallel Processing for Scientific Computing* (2004).

[26] Hysom, D., and Pothen, A. A scalable parallel algorithm for incomplete factor preconditioning. *SIAM Journal on Scientific Computing 22* (2000), 2194–2215.

[27] Mohiyuddin, M., Hoemmen, M., Demmel, J., and Yelick, K. Minimizing communication in sparse matrix solvers. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (New York, NY, USA, 2009), ACM, pp. 1–12.

[28] Nuentsa Wakam, D., Erhel, J., Édouard Canot, and Atenekeng Kahou, G.-A. A comparative study of some distributed linear solvers on systems arising from fluid dynamics simulations. In *Parallel Computing: From Multicores and GPU's to Petascale* (2010), vol. 19 of *Advances in Parallel Computing*, IOS Press, pp. 51–58.

[29] Philippe, B., and Reichel, L. On the generation of krylov subspace bases. Tech. Rep. RR-7099, INRIA, 2009.

[30] Rabenseifner, R., Hager, G., and Jost, G. Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on 0* (2009), 427–436.

[31] Reichel, L. Newton interpolation at leja points. *BIT 30* (1990), 305–331.

[32] Saad, Y. *Iterative Methods for Sparse Linear Systems*, second ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[33] Saad, Y., and Schultz, M. H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing 7*, 3 (1986), 856–869.

[34] Saad, Y., and Sosonkina, M. Distributed schur complement techniques for general sparse linear systems. *SIAM Journal on Scientific Computing 21*, 4 (1999), 1337–1356.

[35] SAMEH, A. Solving the linear leastsquares problem on a linear array of processors. In *High Speed Computer and Algorithm Organization* (1977), D. L. D. Kuck and A. Sameh, Eds., Academic Press, pp. 207–228.

[36] SCHENK, O., AND GÄRTNER, K. Solving unsymmetric sparse systems of linear equations with pardiso. *Journal of Future Generation Computer Systems 20* (2004), 475–487.

[37] SIDJE, R. B. Alternatives for parallel krylov subspace basis computation. *Numerical Linear Algebra with Applications 4*, 4 (1997), 305–331.

[38] SMITH, B., BJϕRSTAD, P., AND GROPP, W. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 1996.

[39] SOUOPGUI, I. Parallélisation d'une double recurrence dans 'gmres'. Master's thesis, University of Yaounde 1, 2005.

[40] WALKER, H. F. Implementation of the 'gmres' method using householder transformations. *SIAM Journal on Scientific and Statistical Computing 9*, 1 (1988), 152–163.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399